AMENDMENTS TO THE SPECIFICATION

Please amend the BRIEF DESCRIPTION OF THE DRAWINGS as follows:

At page 5 of the specification, after line 5, please add the following paragraphs:

FIGURE 4 illustrates a flow diagram of one embodiment of a program that checks for pointer misalignment and appropriately handles a misaligned pointer.

FIGURE 5 illustrates a flow diagram of one embodiment of a program that stores and manipulates pointers in a programming language; and

FIGURE 6 illustrates a flow diagram of another embodiment of a program that stores and manipulates pointers in a programming language.

Please amend the DETAILED DESCRIPTION as follows:

At page 9 of the specification, after line 9, please add the following paragraphs:

FIGURE 4 illustrates a flow diagram 400 of one embodiment of a program that checks for pointer misalignment and appropriately handles a misaligned pointer. According to the illustrated embodiment, the pointer available for use is examined at step 401 to determine if the pointer is a misaligned pointer by checking if type of data or object pointed to is consistent with the data or object type. For example, a pointer is a misaligned pointer if the pointer being used may be defined as pointing to an integer type value, while the pointer actually points to a real value. As another example, the pointer available may be checked to ensure that the address of the pointer is consistent. If the pointer is not a misaligned pointer, the pointer is used as desired by the user's program at step 402.

If, however, the pointer being examined at step 401 is a misaligned pointer, a message is sent to the user (not shown) at step 403, which informs the user that an invalid operation was attempted to be performed on the misaligned pointer. Once this message is displayed to the user, step 404, in one embodiment of the invention, ensures that the execution of the affected operation is halted. Once execution of the operation has been halted at step 404, it may be desirable for execution of the program to be terminated or from the program to continue with reduced functionality. At step 405 the desired next step is determined by the system. For some programs it will be entirely acceptable for the system to wait for the user to correct the inappropriate use of the misaligned pointer before execution of the program is continued. In time critical applications, it would be unacceptable for execution of the program to be halted until the program is fixed. In these cases, a safe recovery state must be identified and programmed into the system for instantiation when an inappropriate use of the misaligned pointer is attempted. In either event, step 406 initiates the desired action.

FIGURE 5 illustrates a flow diagram 500 of one embodiment of a program that stores and manipulates pointers in a programming language. According to the illustrated embodiment, a "safepointer" construct is created at step 502 by instantiating the class "safepointer" 204 that was described with reference to FIGURE 2. The "safepointer" construct is used by a global variable at step 506, where the object was saved as the global variable at step 504, in order to provide information or data to a calling routine, initializing code segment, or other object that is to use a pointer provided by another object. The "safepointer" encapsulates a native pointer within an object that provides automatic pointer checking by performing standard pointer operation upon said object, at step 508. The automatic pointer checking detects and avoids or minimizes misuse of a pointer.

FIGURE 6 illustrates a flow diagram 600 of another embodiment of a program that stores and manipulates pointers in a programming language. According to the illustrated embodiment, a "safepointer" construct is created at step 602 by instantiating the class "safepointer" 204 that was described with reference to FIGURE 2. The "safepointer" construct is called by a function, or subroutine, at step 606, where the object was defined as a function that returns the object "safepointer", at step 604, in order to provide information or data to a calling routine, initializing code segment, or other object that is to use a pointer provided by another object. The "safepointer" encapsulates a native pointer within an object that provides automatic pointer checking by performing standard pointer operation upon said object, at step 608. The automatic pointer checking detects and avoids or minimizes misuse of a pointer.